Using ILP to learn a domain theory in the form of a FSA

Maria LIAKATA and Stephen PULMAN
Centre for Linguistics and Philology
Walton Street
University of Oxford
U.K.,
maria.liakata@clg.ox.ac.uk, stephen.pulman@clg.ox.ac.uk

June 13, 2005

Abstract

This paper describes a method for inducing a domain theory from a corpus of parsed sentences by means of ILP techniques. A 'domain theory' in the current context stands for a collection of facts and generalisations or rules which describe entities and relations between entities within a domain of interest. As language users, we implicitly draw on such theories in various disambiguation tasks, such as anaphora resolution and prepositional phrase attachment, or to draw inferences. Formal encodings of domain theories can be used for this purpose in natural language processing but they may also be objects of interest in their own right, that is, as the output of a knowledge discovery process. The patterns learnt are represented as FSAs, providing a graphical representation and a more compact format than the original symbolic rules. The approach is generizable to different domains provided it is possible to get logical forms for the text in the domain.

1. Introduction

It is an old observation that in order to choose the correct reading of an ambiguous sentence, we need a great deal of knowledge about the world. Moreover, world knowledge is crucial to all types of linguistic disambiguation and reasoning, as reflected in work such as [Hobbs et al.1993], where hand-coded pragmatic inference rules are used for linguistic interpretation in naval operation and terrorist reports. Rather than constructing such rules manually, which is time consuming and difficult to replicate, the current paper proposes a method for learning automatically from text a set of domain rules describing associations between two or more verbs and their respective arguments. Note that this is a different goal from merely obtaining selectional preferences for verbs as we are primarily interested in detecting more long range dependencies between verbs.

Appearing in *Proceedings of the 4^{th} Learning Language in Logic Workshop (LLL05)*, Bonn, Germany, 2005.

2. Some background

[Pulman2000] describes a method for obtaining a theory for prepositional phrase disambiguation. This approach inverts the observation that disambiguation decisions depend on knowledge of the world and showed that it is possible to learn a simple domain theory from a disambiguated corpus by capitalising on the information tacitly contained in the disambiguation decisions. Ambiguous sentences from a subset of the ATIS (air travel information service) corpus [Doddington and Godfrey1990] were annotated so as to indicate the preferred reading, e.g.

```
[do,they,[serve,a,meal],on,
[the,flight,from,san_francisco,to,atlanta]]
```

The 'good' and the 'bad' parses were used to produce simplified first order logical forms representing the semantic content of the various readings of the sentences. The 'good' readings were used as positive evidence, and the 'bad' readings (or more accurately, the bad parts of some of the readings) were used as negative evidence. Next the Inductive Logic Programming algorithm, Progol [Muggleton1995], was used to learn a theory of prepositional relations in this domain: i.e. what kinds of entities can be in these relations, and which cannot.

Among others generalisations like the following were obtained (all variables are implicitly universally quantified):

$$\begin{array}{ll} fare(A) \wedge airline(B) & \rightarrow on(A,B) \\ meal(A) \wedge flight(B) & \rightarrow on(A,B) \end{array}$$

This domain theory was then used successfully in disambiguating a small held-out section of the corpus, by checking for consistency between logical forms and domain theories.

While the numbers of sentences involved in that experiment were too small for the results to be statistically meaningful, the experiment proved that the method works in principle. Moreover, the results of the theory induction process are perfectly comprehensible - the outcome is a theory with some logical structure, rather than a black box.

3. Current Approach

The current paper seeks to extend this approach and at the same time derive less specialised rules that can be useful in a variety of tasks ranging from different types of linguistic disambiguation to reasoning (e.g. in question-answering). The method requires a fully parsed corpus with corresponding logical forms. We have experimented with larger datasets, using the Penn Tree Bank [Marcus et al.1994] since the syntactic annotations for sentences given there are intended to be complete enough for semantic interpretation, in principle, at least.

In practice, [Liakata and Pulman2002] report, it is by no means easy to do this. It is possible to recover partial logical forms from a large proportion of the treebank, but these are not complete or accurate enough to simply replicate the ATIS experiment. In the work reported here, we selected about 40 texts containing the verb 'resign', all reporting, among other things, 'company succession' events, a scenario familiar from the Message Understanding Conference (MUC) task [Grishman and Sundheim1995]. The texts amounted to almost 4000 words in all. Then we corrected and completed some automatically produced logical forms by hand to get a fairly full representation of the meanings of these texts (as far as is possible in first order logic). We also resolved by hand some of the simpler forms of anaphoric reference to individuals to simulate a fuller discourse processing of the texts.

To give an example, a sequence of sentences like:

J.P. Bolduc, vice chairman of W.R. Grace & Co. (...) was elected a director. He succeeds Terrence D. Daniels,... who resigned.

was represented by the following sequence of literals:

```
verb(e1,elect).
funct_of('J.P._Bolduc',x1).
...
subj(e1,unspecified).
obj(e1,x1).
description(e1,x1,director,de1).
verb(e5,succeed).
subj(e5,x1).
funct_of('Terrence_D._Daniels',x6).
obj(e5,x6).
verb(e4,resign).
subj(e4,x6).
```

The representation is a little opaque, for implementation reasons relating to the input settings of the learning mechanism. The above example can be paraphrased as follows: there is an event, e1, of electing, the subject of which is unspecified, and the object of which is x1. x1 is characterised as 'J P Bolduc', and e1 assigns the description de1 of 'director' to x1. There is an event e5 of succeeding, and x1 is the subject of that event. The object of e5 is x6, which is characterised as Terrence D Daniels. There is an event e4 of resigning and the subject of that event is x6.

The reason for all this logical circumlocution is that we are trying to learn a theory of the 'verb' predicate, in particular we are interested in relations between different verbs as well as associations between verbs and their argument since these may well be indicative of causal or other regularities that should be captured in the theory of the company succession domain. If the individual verbs were represented as predicates rather than arguments of a 'verb' predicate we would not be able to generalise over them: we are restricted to first order logic, and this would require higher order variables.

We also need to add some background knowledge. We assume a fairly simple flat ontology. It is also possible to work with hierarchical ontologies, for example WordNet. Indeed, in a version of our experiment we obtained all hypernym chains for verbs and configured the learning algorithm to add a verb concept to each iteration, from most general to specific, until the pattern would fail to exceed a certain threshold. However, this approach led to overly general, uninformative patterns. For the purpose of this experiment, it was not deemed necessary to repeat the same process with verb arguments.

Entities were assigned to classes semi-automatically, using clustering techniques described in [Liakata2004] followed by manual adjustment. The latter involved merging the 32 classes resulting from the automatic clustering into 11 principal categories, namely the following:

company, financial instrument, financial transaction, location, money, number, person, company position, product, time, and unit (of organisation).

As with the 'verb' predicate, the representation has these categories as an argument of a 'class' predicate to enable generalisation:

```
class(person,x1).
class(company,x3).
etc.
```

Ideally, to narrow down the hypothesis space for ILP, we need some negative evidence. In the Penn Tree Bank, though, only the good parse is represented. There are several possible ways of obtaining negative data: one could use a parser trained on the Tree Bank to reparse sentences and recover all the parses. However, there still remains the problem of recovering logical forms from 'bad' parses. An alternative would be to use a kind of 'closed world' assumption: take the set of predicates and arguments in the good logical forms, and assume that any combination not observed is actually impossible. One could generate artificial negative evidence this way but one risks overgenerating and rejecting otherwise plausible combinations.

Alternatively, one can try learning from positive only data. The ILP systems Progol [Muggleton1995] and Aleph [Srinivasan1999] are able to learn from positive only data, with the appropriate settings. Likewise, so-called 'descriptive' ILP systems like WARMR [Dehaspe1998] do not always need negative data: they are in effect data mining engines for first order logic, learning generalisations and correlations in some set of data.

4. Learning rules for company succession events

We found that the most successful method, given the absence of negative data, was to use WARMR to learn association rules from the positive data. WARMR is very closely related to the levelwise [Mannila and Toivonen1997] and the APRIORI algorithms [Agrawal et al.1996]. The latter assume a lattice of data, denoting relations between binary valued attributes and the algorithm performs a general-to-specific, breadth-first search, looking at one level of the data lattice at a time. The patterns are specialised at each level, by the addition of an extra attribute-value pair. The method iterates between candidate generation and candidate evaluation. The algorithm implemented in WARMR is a version of the levelwise algorithm where binary attributes have been replaced by predicate calculus literals.

The input to WARMR consists of the evidence (sentences) represented in terms of models, where each model can span over several sentences, thus increasing the possibilities of associations between verbs, which constitute our main point of interest. The language bias is made available as a set of type and mode declarations, called rmodes. The rmodes define a set of literal lists, which are interpreted by the system. WARMR rmodes are interpreted as queries and checked against the evidence at each iteration of the algorithm.

Examples of rmodes are:

```
%1
```

```
rmode(3:verb(-Sub,\Event,#1)).

%2
rmode((attr(#1,+Event,\ID), class(#1,ID))).
```

The first rmode declaration says that a likely pattern is expected to contain up to three verbs. The '\ Event' variable means that the verb introduced should correspond to a new event variable whose value is different from others of the same type. The '#1' symbol is a way of denoting that the corresponding slot is a constant, generated from the data. The second rmode says that a likely pattern should include an attribute along with its class information.

As with all types of association rule learning, WARMR produces a huge number of rules, of varying degrees of coverage. We spent some time writing filters to narrow down the output to something useful. Such filters consist of constraints ruling out patterns that are definitely not useful, for example patterns containing a verb but no arguments or attributes. An example of such a restriction is provided below:

```
pattern_constraint(Patt):-
     member(verb(_,E,_A,_,_),Patt),
     (member(attr(_,E,Attr),Patt)
      ->
      \+constraint_on_attr(Patt,Attr)).
```

If $pattern\_constraint/1$ succeeds for a pattern Patt, then Patt is discarded. Basically, this says that a rule isn't useful unless it contains a verb and one of its attributes that satisfies a certain constraint. A constraint might be of the following form:

```
constraint_on_attr(Patt, Attr) :-
      member(class(_,Attr), Patt).
```

The above states that there should be a classification of the attribute Attr present in the rule. A useful pattern Patt will satisfy such constraints.

Some of the filtered output, represented in a more readable form compatible with the examples above are as follows (note that the first argument of the verb/2 predicate refers to an event):

Companies report financial transactions:

$subj(B,C) \wedge obj(B,D) \wedge class(fin\_tran,D) \wedge class(company,C) \rightarrow verb(B,report)$

Companies acquire companies:

$subj(B,C) \wedge obj(B,D) \wedge class(company,D) \wedge class(company,C) \rightarrow verb(B,acquire)$

Companies are based in locations:

$obj(A,C) \wedge class(company,C) \wedge in(A,D) \wedge class(location,D) \rightarrow verb(A,base)$

If person C succeeds person E, then someone has elected person C:

$obj(A,C) \wedge class(person,C) \wedge verb(D,succeed) \wedge$ subj(D,C) $\wedge obj(D,E) \wedge class(person,E) \rightarrow verb(A,elect)$

If someone elects person C, and person D resigns, then C succeeds D:

$subj(G,C) \wedge verb(A,elect) \wedge obj(A,C) \wedge class(person,C) \wedge verb(E,resign) \wedge subj(E,D) \wedge class(person,D) \rightarrow verb(G,succeed)$

While there are many other rules learned that are less informative than this, the samples given here are true generalisations about the type of events described in these texts: unremarkable, perhaps, but characteristic of the domain. It is noteworthy that some of them at least are very reminiscent of the kind of templates constructed for Information Extraction in this domain, suggesting a possible further use for the methods of theory induction described here.

5. Representing the Output

Each of the numerous patterns resulting from WARMR consists of a list of frequently associated predicates, found in the flat quasi-logical forms of the input sentences. An example of such a pattern is provided by the following:

```
freq(6,[verb(A,B,elect,p,d),
        verb(C,D,succeed,p,d),
        attr(subj,B,unspecified),
        attr(obj,D,E),class(cperson,E),
        attr(subj,D,F),class(cperson,F),
```

```
    attr(obj,B,F)],
  0.1463).
```

The first argument of the predicate $freq/3$ shows the level of the algorithm at which the pattern/query was acquired [Dehaspe1998]. The fact that the pattern was acquired at the sixth level means it was created during the sixth iteration of the algorithm trying to satisfy the constraints input as settings to the system. This pattern satisfied four constraints, two of them twice[1]. The second argument of $freq/3$ is the query itself and the third is its frequency. What is meant by frequency of the query in this instance is the number of times it succeeds (i.e. the number of training examples it subsumes), divided by the number of training examples. To illustrate the meaning of such a pattern one needs to reconstruct the predicate-argument structures while maintaining the flat format. Thus, the above pattern is converted to the following:

```
list(529,0.1463,[elect(A,B,C),
                 cperson(C),
                 succeed(D,C,E),
                 cperson(E)]).
```

It is now easier to understand the pattern as :'A person C who is elected succeeds a person E'. However, it is still not straightforward how one can evaluate the usefulness of such patterns or indeed how one can incorporate the information they carry into a system for disambiguation or reasoning. This problem is further aggravated by the large number of patterns produced. Even after employing filters to discard patterns of little use, for example ones containing a verb but no classification of its arguments, over 26,000 of them were obtained. Experimenting with a more restrictive language bias is an option but our experience was that there is a significant trade off between complex rmodes and algorithm efficiency. We found that it was better to relax the language settings and allow WARMR to perform more iterations, while at the same time requiring more extensive filtering of the output.

The large size of the output is mainly due to the fact that many patterns are overly general: the training set consists of only 372 verb predicates and a total of 436 clauses. Such overgeneration is a well known problem of data mining algorithms and requires sound criteria for filtering and evaluation. Most of the patterns generated are in fact variants of a much smaller group of patterns. The question then arises of how it is possible to merge them so as to obtain a small number of core patterns, representative of the knowledge obtained from the training set. Representing the patterns in a more compact format also facilitates evaluation either by a human expert or through incorporation into a pre-existing system to measure improvement in performance.

[1] There are eight literals in the pattern, even though it was obtained in round 6. This is because the rmode satisfied at rounds 4 and 5 adds two literals simultaneously.

6. FSA conversion

Given the large amount of shared information in these outputs, we decided to try to represent it as a set of Finite State Automata, where each transition corresponds to a literal in the original clauses[2] Since all the literals in the raw output are simply conjoined, the interpretation of a transition is simply that if one literal is true, the next one is also likely to be true. Our aim was to be able to use standard FSA minimisation and determination algorithms [Aho et al.1986],[Aho et al.1974] to reduce the large set of overlapping clauses to something manageable and visualisable, and to be able to use the frequency information given by WARMR as the basis for the calculation of weights or probabilities on transitions.

To convert our patterns into FSAs (and in particular recognizers), we used the package FSA Utilities (version FSA6.2.6.5)[van Noord2002], which includes modules for compiling regular expressions into automata (recognizers and transducers) by implementing different versions of minimisation and determinisation algorithms. The package also allows operations for manipulating automata and regular expressions such as composition, complementation etc. As the FSA Utilities modules apply to automata or their equivalent regular expressions, the task required converting the patterns into regular expressions. To do this we treat each literal as a symbol. This means each verb and attribute predicate with its respective

[2] One possibility would have been to employ Galois lattices to merge together patterns subsuming each other. In that case single predicates or tuples would be used as descriptors-attributes of the patterns-objects. However, weighted FSAs present a more straightforward way of designating the dependencies between patterns and preserve the ordering of literals in the pattern.

arguments is taken to denote a single symbol. The literals are implicitly conjoined and thus ordering does not matter. Thus we chose to impose an ordering on patterns, whereby the main verb appears first, followed by predicates referring to its arguments. Any other verbs come next, followed by predicates describing their arguments. This ordering has the advantage over alphanumeric ordering that it allows filtering out alphabetic variants of patterns where the predicates referring to the arguments of a verb precede the verb and the variables are thus given different names which results in different literals. This ordering on patterns is useful as it allows common prefixes to be merged during minimisation. Since variable names play an important role in providing co-indexation between the argument of a verb and a property of that argument, designated by another predicate, terms such as '$elect(A, B, C)$' and '$elect(D, E, F)$' are considered to be different symbols. Thus a pattern like:

```
list(768,0.07,[elect(A,B,C),cperson(C),
               chairman(C,D),old(C,E,F),
               of(D,G),ccompany(G)]).
```

was converted to the regular expression:

```
macro(x768,['elect(A,B,C)',
            'cperson(C)',
            'chairman(C,D)',
            'old(C,E,F)',
            'of(D,G)',
            'ccompany(G)']).
```

The first argument of the $macro/2$ predicate is the name of the regular expression whereas the second argument states that the regular expression is a sequence of the symbols 'elect(A,B,C)','cperson(C)','chairman(C,D)' and so on. Finally, the entire WARMR output can be compiled into an FSA as the regular expression which is the union of all expressions named via an xnumber identifier. This is equivalent to saying that a pattern can be any of the xnumber patterns defined.

We took all the patterns containing 'elect' as the main verb and transformed them to regular expressions, all of which started with 'elect(A,B,C)'. We then applied determinisation and minimisation to the union of these regular expressions. The result was an automaton of 350 states and 839 transitions, compared to an initial 2907 patterns.

However, an automaton this size is still very hard to visualize. To circumvent this problem we made use of the properties of automata and decomposed the regular expressions into subexpressions that can then be conjoined to form the bigger picture. Patterns containing two and three verbs were written in separate files and each entry in the files was split into two or three different segments, so that each segment contained only one verb and predicates referring to its arguments. Therefore, an expression such as:

```
macro(x774,[elect(A,B,C),cperson(C),
            resign(D,E,F),cperson(E),
            succeed(G,C,E)]).
```

was transformed into:

```
macro(x774a,['elect(A,B,C)',
             'cperson(C)']).
macro(x774b,['resign(D,E,F)',
             'cperson(E)']).
macro(x774c,['succeed(G,C,E)']).
```

One can then define the automaton xpression1, consisting of the union of all first segment expressions, such as x774a, the automaton resign2, consisting of all expressions where resign is the second verb and succeed3. The previous can be combined to form the automata $[xpression1, resign2]$ or $[xpression1, resign2, succeed3]$ and so on. The automaton $[xpression1, resign2]$ which represents 292 patterns, has 32 states and 105 transitions and is much more manageable.

7. Adding weights

The FSA rules derived from the WARMR patterns would be of more interest if weights were assigned to each transition, indicating the likelihood of any specific path/pattern occurring. For this we needed to obtain weights, equivalent to probabilities for each literal/predicate-argument term. Such information was not readily available to us. The only statistics we have correspond to the frequency of each entire pattern, which is defined as:

$$Freq = \frac{\text{number of times the pattern matched the training data}}{\text{number of examples in the training set}}$$

We took this frequency measure as the probability of patterns consisting of single predicates (e.g. 'elect(A,B,C)', which is equivalent to 'B elects C') whereas the probabilities of all other pattern constituents have to be conditioned on the probabilities of terms preceding them. Thus, the probability of 'cperson(C)', given 'elect(A,B,C)' is defined by the following:

$$P('cperson(C)'|'elect(A,B,C)') = \frac{P('elect(A,B,C)','cperson(C)')}{P('elect(A,B,C')}$$

where $P('elect(A,B,C)','cperson(C))'$ is the frequency of the pattern $['elect(A,B,C)','cperson(C)']$ and $P('elect(A,B,C)')$ is defined as:

$$P('elect(A,B,C)') = \sum_X P('elect(A,B,C)',X)$$

That is, the probability of $P('elect(A,B,C)')$ is the sum of all the probabilities of the patterns that contain 'elect(A,B,C)' followed by another predicate.

In principle the frequency ratios described above are probabilities but in practice, because of the size of the dataset, they may not approximate real probabilities. Either way they are still valid quantities for comparing the likelihood of different paths in the FSA.
Having computed the conditional probabilities/weights for all patterns and constituents, we normalized the distribution by dividing each probability in a distribution by the total sum of the probabilities. This was necessary in order to make up for discarded alphabetic variants of patterns. We then verified that the probabilities summed up to 1. To visualise some of the FSAs (weighted recognizers) we rounded the weights to the second decimal digits and performed determinization and minimization as before. Rules obtained can be found in Figures 1 and 2 (see figures on last page):

The automaton of Figure 1 incorporates the following rules:

1. 'If a person C is elected, another person E has resigned and C succeeds E'
2. 'If a person C is elected director then another person F has resigned and C succeeds F'
3. 'If a person C is elected and another person E pursues (other interests) C succeeds E'

The automaton of Figure 2 provides for rules such as: 'If a person is elected chairman of a company E then C succeeds another person G'.

At each stage, thanks to the weights, it is possible to see which permutation of the pattern is more likely.

8. Related Work

Rules such as the above express causality and interdependence between semantic predicates, which can be used to infer information for various linguistic applications. The idea of deriving inference rules from text has been pursued in [Lin and Pantel2001] as well, but that approach differs significantly from the current one in that it is aimed mainly at discovering paraphrases. In their approach text is parsed into paths, where each path corresponds to predicate argument relations and rules are derived by computing similarity between paths. A rule in this case constitutes an association between similar paths. This is quite different to the work currently presented, which provides more long range causality relations between different predicates, which may not even occur in adjacent sentences in the original texts. Other approaches such as [Collin et al.2002] also aim to learn paraphrases for improving a Question-Answering system. Our work is perhaps more closely related to the production of causal networks as in [Subramani and Cooper1999], where the goal is to learn interdependency relations of medical conditions and diseases. In their work the dependencies only involve key words, but we believe that our techniques could be applied to similar biomedical domains to discover causal theories with richer inferential structure.

9. Conclusions & Future Work

We have shown that it is possible to induce logically structured inference rules from parsed text. We have also shown that by using FSA techniques it is possible to construct a weighted automaton for the representation of rules/patterns generated via a knowledge mining process. This enables merging together permutations of the same pattern and facilitates human evaluation of the pattern. Furthermore, the fact that we have learned what is in effect a simple probabilistic graphical model means that we can now produce representations of this knowledge suitable for more robust inference methods of the type that we can deploy to aid reasoning and disambiguation tasks. The current approach is also interesting as a means of obtaining new knowledge, through previously undetected associations (e.g. if applied to medical texts). Patterns acquired would then serve the purpose of bringing new insight to the knowledge expertise in question.

10. Acknowledgements

We would particularly like to thank Ashwin Srinivasan (IBM, New Delhi), Steve Moyle (Oxford), and James Cussens (York) for their help with Aleph and Jan Struyf, Hendrik Blockeel and Jan Ramon (K.U. Leuven), for their generous help with WARMR.

References

[Agrawal et al.1996] R. Agrawal, H. Mannil, R. Srikant, H. Toivonen, and A.I. Verkamo. 1996. Fast discovery of association rules. In P. Smyth U.M. Fayyad, G. Piatetsky-Shapiro and R. Uthurusamy, editors, *Adevances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press, Menlo Park, CA.

[Aho et al.1974] A.H. Aho, J.E. Hopcroft, and J.D. Ullman. 1974. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company.

[Aho et al.1986] A.H. Aho, R. Sethi, and J.D. Ullman. 1986. *Compilers - Principles, Techniques, and Tools*. Addison-Wesley, Reading, Massachusetts, USA.

[Collin et al.2002] O. Collin, F. Duclaye, and F. Yvon. 2002. Learning Paraphrases to Improve a Question-Answering System. *staff.science.uva.nl/ mdr/NLP4QA/10duclaye-et-al.pdf*.

[Dehaspe1998] Luc Dehaspe. 1998. *Frequent Pattern Discovery in First-Order Logic*. Ph.D. thesis, Katholieke Universiteit Leuven.

[Doddington and Godfrey1990] G. Doddington and C.H.J. Godfrey. 1990. The ATIS Spoken Language Systems Pilot Corpus. In *Speech and Natural Language Workshop*, Hidden Valley, Pennsylvania.

[Grishman and Sundheim1995] R. Grishman and B. Sundheim. 1995. “Message Understanding Conference-6: A Brief History”. *www.cs.nyu.edu/cs/projects/proteus/muc/muc6-history-coling.ps*.

[Hobbs et al.1993] J. Hobbs, M. Stickel, D. Appelt, and P. Martion. 1993. Interpretation as abduction. *Artificial Intelligence*, 63:69–142.

[Liakata and Pulman2002] M. Liakata and S. Pulman. 2002. From Trees to Predicate-Argument Structures. In *International Conference for Computational Linguistics (COLING)*, pages 563–569, Taipei, Taiwan.

[Liakata2004] M. Liakata. 2004. Inducing domain theories. D.Phil. thesis.

[Lin and Pantel2001] D. Lin and P. Pantel. 2001. Dirt-Discovery of Inference Rules from Text. In *In ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 323–328.

[Mannila and Toivonen1997] H. Mannila and H. Toivonen. 1997. Levelwise Search and Borders of Theories in Knowledge Discovery. *Data Mining Knowledge Discovery*, 1(3):241–258.

[Marcus et al.1994] M. Marcus, G. Kim, M. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. 1994. The Penn Treebank: Annotating predicate argument structure. In *ARPA Human Language Technology Workshop*.

[Muggleton1995] Stephen Muggleton. 1995. Inverse Entailment and Progol. *New Generation Computing, special issue on Inductive Logic Programming*, 13(3-4):245–286.

[Pulman2000] Stephen Pulman. 2000. Statistical and Logical Reasoning in Disambiguation. *Philosophical Transactions of the Royal Society*, 358 number 1769:1267–1279.

[Srinivasan1999] Ashwin Srinivasan. 1999. “the Aleph Manual”. *www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/*.

[Subramani and Cooper1999] M. Subramani and G.F. Cooper. 1999. Causal Discovery from Medical Textual Data. *http://www.amia.org/pubs/symposia/D200558.PDF*.

[van Noord2002] Gertjan van Noord. 2002. FSA6 Reference Manual. *http://odur.let.rug.nl/ vannoord/Fsa/*.

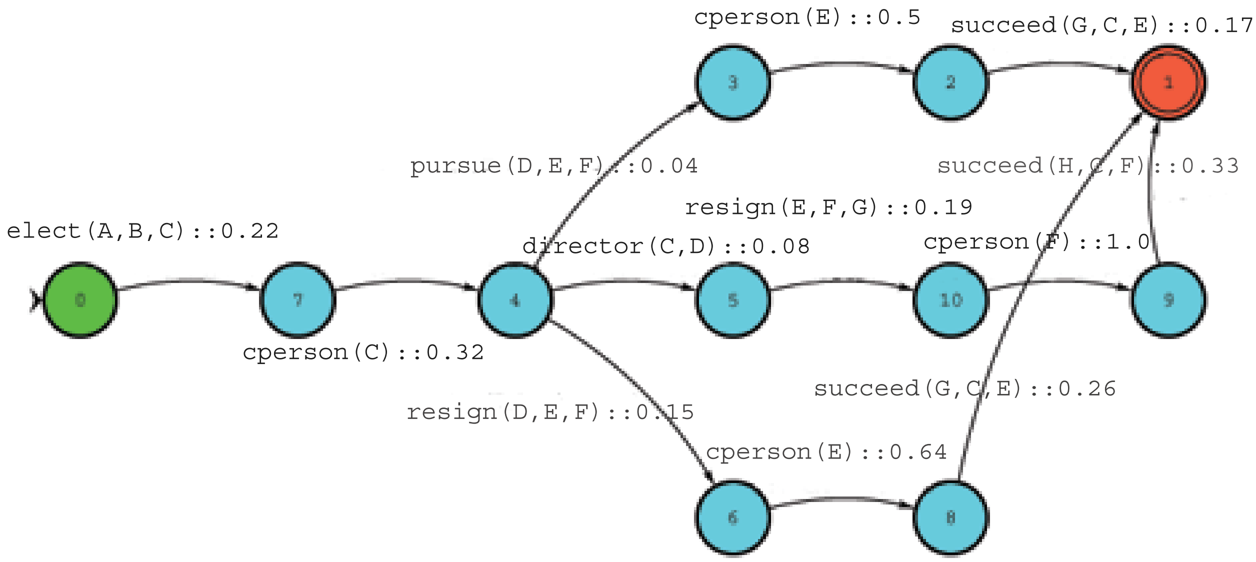

Figure 1. The more likely path in this FSA segment is given by the choice of $resign(D, E, F) : 0.15$, followed by $cperson(E) : 0.64$ and finally $succeed(G, C, E) : 0.26$. This can be interpreted as follows: 'If a person C is elected, another person E has resigned and C succeeds E'

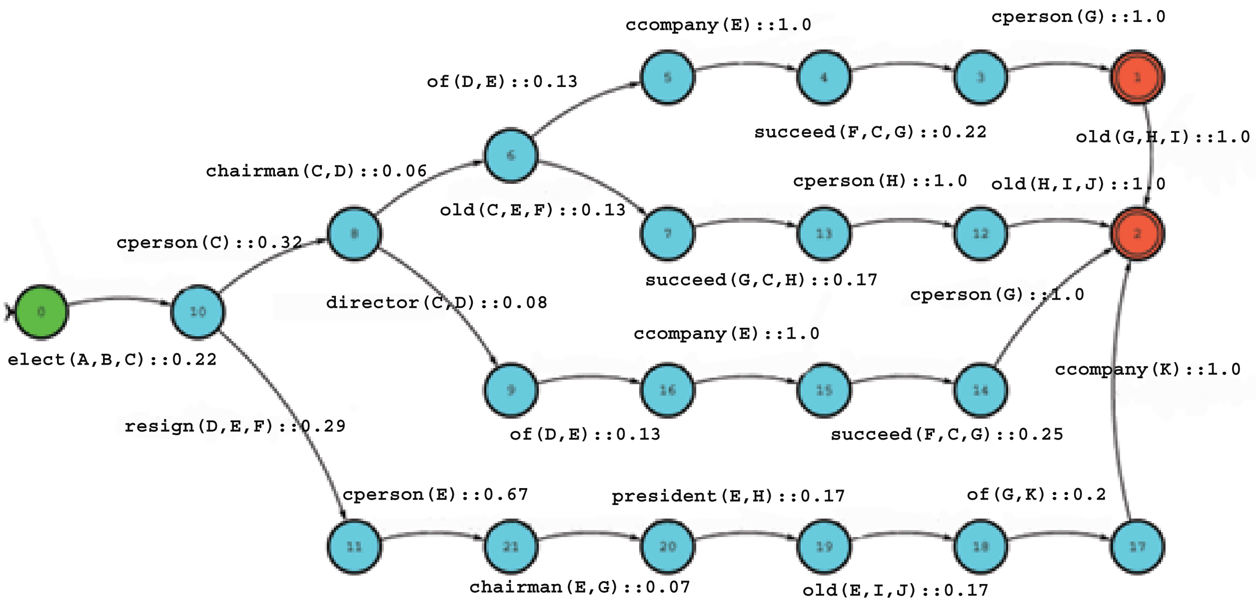

Figure 2. Here the more likely path is provided by the sequence: $cperson(C) : 0.32$, $director(C, D) : 0.08$, $of(D, E) : 0.13$, $company(E) : 1$, $succeed(F, C, G) : 0.25$, $cperson(F) : 1$. This can be read as: 'If a person C is elected director of a company E then C succeeds another person G'.

Notice the above illustrate only parts of the FSAs, which justifies why the probabilites of arcs leaving a node don't add up to 1